

DAFNY

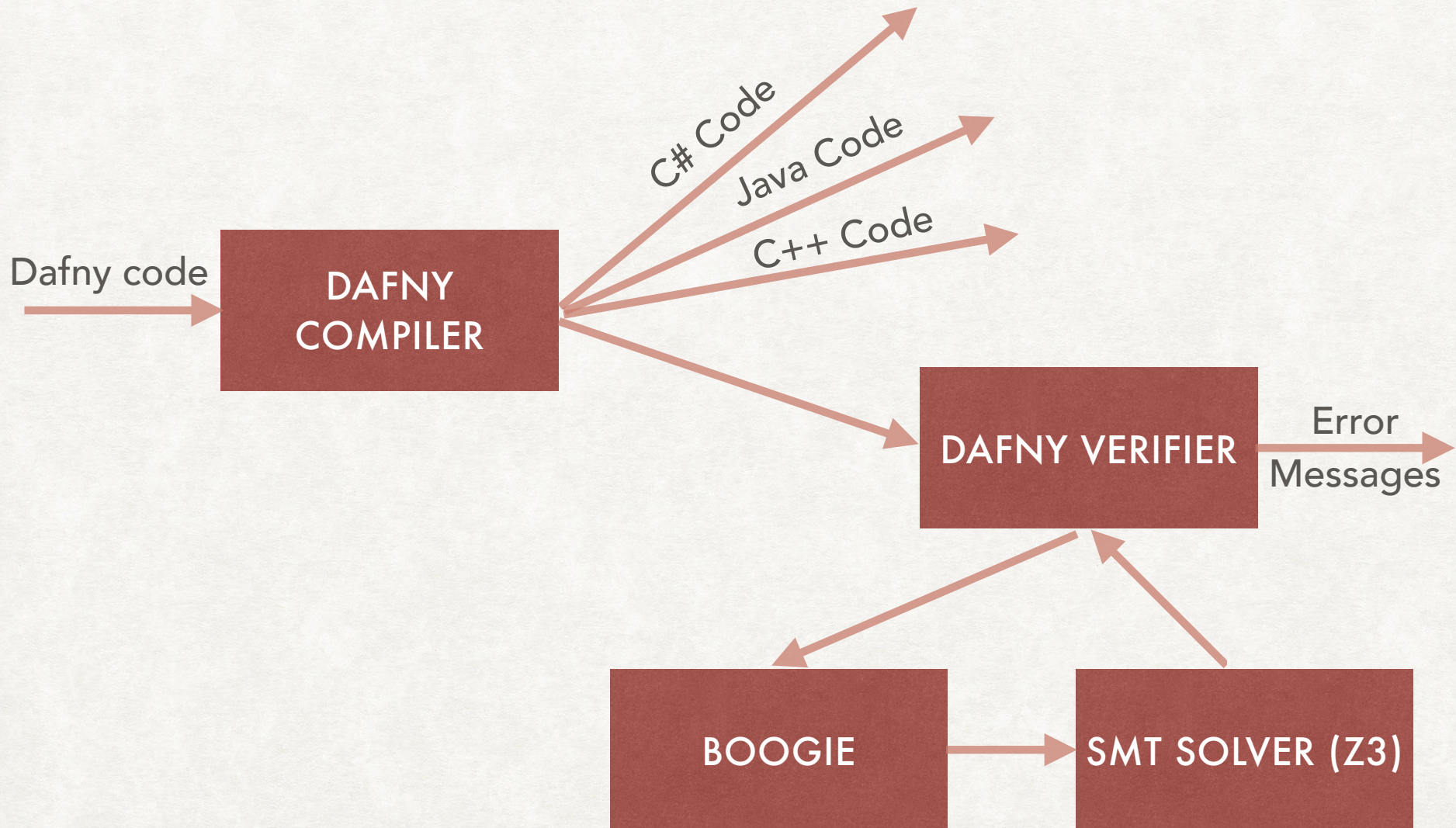
INTRODUCTION

- Dafny is a programming language, verifier and compiler, all rolled together into one.
- Dafny uses SMT solver (Z3) to automatically prove correctness.
- Highly sophisticated specification language
 - Pre-conditions and Post-conditions for methods
 - Loop Invariants
 - Assertions
 - Lemmas and Theorems

DAFNY - PROGRAMMING LANGUAGE

- Dafny hosts two separate sub-languages
- Language for the executable code
 - Imperative, Object-oriented
 - Methods, loops, if-then-else, arrays, classes, inductive datatypes,...
- Specification Language
 - Functional
 - Pure functions, predicates, algebraic datatypes, sets, sequences, lemmas...

DAFNY TOOL



DAFNY BASICS: METHODS

- Parameters and return values must be typed.
- Return values are named.
- Multiple return values are also allowed.

```
method Abs(x: int) returns (y: int)
{
  ...
}
```

```
method MultipleReturns(x: int, y: int)
returns (more: int, less: int)
{
  ...
}
```

DAFNY BASICS: METHODS

```
method MultipleReturns(x: int, y: int)
returns (more: int, less: int)
{
    more := x + y;
    less := x - y;
}
```

- To return a value, assign it to the named return variable.
 - In case of a single return value, the 'return' statement can also be used.
- Assignments use ':='.
- Compound statements such as if..then..else, while loops are available.

DAFNY BASICS: PRE AND POST-CONDITIONS

```
method MultipleReturns(x: int, y: int)
  returns (more: int, less: int)
    ensures less < x
    ensures x < more
  {
    more := x + y;
    less := x - y;
  }
```

- Post-conditions specified using 'ensures'.
- Are the post-conditions satisfied in the above example?

DAFNY BASICS: PRE AND POST-CONDITIONS

```
method MultipleReturns(x: int, y: int)
  returns (more: int, less: int)
    requires y > 0
    ensures less < x
    ensures x < more
  {
    more := x + y;
    less := x - y;
  }
```

- Post-conditions specified using 'ensures'.
 - Are the post-conditions satisfied in the above example?
- Pre-conditions specified using 'requires'
 - Dafny will output 'Verified'

DAFNY BASICS: ASSERTIONS

- Local variables declared using 'var'.
- Methods calls are not allowed inside assert conditions.
- Will the assert in 'Testing' method succeed?

```
method Abs(x: int) returns (y: int)
  ensures y >= 0
{
  if x < 0
    { return -x; }
  else
    { return x; }
}
method Testing()
{
  var v := Abs(3);
  assert v == 3;
}
```

DAFNY BASICS: ASSERTIONS

- Local variables declared using 'var'.
- Methods calls are not allowed inside assert conditions.
- Will the assert in 'Testing' method succeed?

```
method Abs(x: int) returns (y: int)
  ensures y >= 0
  ensures x >= 0 ==> y == x
  ensures x < 0 ==> y == -x
{
  if x < 0
    { return -x; }
  else
    { return x; }
}
method Testing()
{
  var v := Abs(3);
  assert v == 3;
}
```

DAFFY BASICS: FUNCTIONS

```
function abs(x: int): int
{
  if x < 0 then -x else x
}
```

- Pure mathematical functions
 - Cannot write to memory
 - Body is a single expression
 - Single return value
 - Not compiled and executed
- Can only be used inside pre-conditions, post-conditions, assertions and loop invariants

DAFNY BASICS: FUNCTIONS

```
function abs(x: int): int
{
  if x < 0 then -x else x
}
method Abs(x: int) returns (y: int)
  ensures y == abs(x)
{
  if x < 0
    { return -x; }
  else
    { return x; }
}
method Testing()
{
  var v := Abs(3);
  assert v == 3;
}
```

DAFFY BASICS: LOOP INVARIANTS

```
var i := 0;  
while i < n  
  invariant 0 <= i  
  {  
    i := i + 1;  
  }
```

- Specified using the keyword 'invariant'
 - Must hold upon entering the loop
 - After every iteration of the loop body
 - Dafny will automatically check the conditions

DAFFY BASICS: LOOP INVARIANTS

```
method ComputeFib(n: nat) returns (b: nat)  
  ensures b == fib(n)
```

```
{  
  if n == 0 { return 0; }  
  var i: int := 1;  
  var a := 0;  
    b := 1;  
  while i < n  
    invariant 0 < i <= n  
    invariant a == fib(i - 1)  
    invariant b == fib(i)  
  {  
    a, b := b, a + b;  
    i := i + 1;  
  }  
}
```

```
function fib(n: nat): nat  
{  
  if n == 0 then 0 else  
  if n == 1 then 1 else  
    fib(n - 1) + fib(n - 2)  
}
```

DAFFY BASICS: TERMINATION

- Dafny will try to prove that every loop terminates
 - Reports an error if it can't prove termination.
- Dafny will guess an expression which decreases with every iteration and has a lower bound.
 - This can be explicitly specified using the 'decreases' annotation.
 - Use the annotation 'decreases *' if we don't want Dafny to prove termination.

```
while 0 < i
  invariant 0 <= i
  decreases i
{
  i := i - 1;
}
```

DAFFY BASICS: TERMINATION

- Dafny will try to prove that every loop terminates
 - Reports an error if it can't prove termination.
- Dafny will guess an expression which decreases with every iteration and has a lower bound.
 - This can be explicitly specified using the 'decreases' annotation.
 - Use the annotation 'decreases *' if we don't want Dafny to prove termination.

```
while 0 < i
  invariant 0 <= i
  decreases i
{
  i := i - 1;
}
```

```
while i < n
  invariant 0 <= i <= n
  decreases n - i
{
  i := i + 1;
}
```


DAFFY BASICS: ARRAYS

- Built in data type 'array<T>'
 - Also built in field 'Length'.
- Dafny will automatically check that arrays are accessed within the bounds.

```
method M(x: int)
```

```
{
```

```
  var a := new int[10];
```

```
  var b := a[x]; //Error
```

```
  if 0 <= x < 10
```

```
    { b := a[x]; //OK }
```

```
}
```

```
method M(x: int, c: array<int>)
```

```
  requires 0 <= x < 10
```

```
{
```

```
  var a := new int[10];
```

```
  var b := a[x]; //OK
```

```
  If 0 <= x < 10
```

```
    { b := c[x]; //Error }
```

```
}
```

DAFNY BASICS: QUANTIFIERS

```
method Find(a: array<int>, key: int) returns (index: int)
  ensures 0 <= index ==> index < a.Length && a[index] == key
  ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != key
{
  index := 0;
  while index < a.Length
    invariant 0 <= index <= a.Length
    invariant forall k :: 0 <= k < index ==> a[k] != key
    {
      if a[index] == key { return; }
      index := index + 1;
    }
  index := -1;
}
```

- Quantifiers can be used in pre/post-conditions, assertions, invariants using the 'forall' keyword.

DAFNY BASICS: LEMMAS

```
method ComputePow2(n: nat) returns (p:nat)
  ensures p = pow2(n)
{
  if n = 0
  { p := 1; }
  else if n % 2 = 0
  { p := ComputePow2(n / 2);
    p := p * p; }
  else
  { p := ComputePow2(n-1);
    p := 2 * p;
  }
}

function pow2(n: int): int
  requires 0 <= n
{
  if n == 0 then 1
  else
    2*pow2(n-1)
}
```

DAFNY BASICS: LEMMAS

```
method ComputePow2(n: nat) returns (p:nat)
  ensures p = pow2(n)
{
  if n = 0
  { p := 1; }
  else if n % 2 = 0
  { p := ComputePow2(n / 2);
    p := p * p;
    Lemma(n); }
  else
  { p := ComputePow2(n-1);
    p := 2 * p;
  }
}
```

```
function pow2(n: int): int
  requires 0 <= n
{
  if n == 0 then 1
  else
    2*pow2(n-1)
}
```

```
Lemma Lemma(n : nat)
  requires n % 2 == 0
  ensures pow2(n) == pow2(n/2)
  * pow2(n/2)
{
  if n != 0 then Lemma(n-2);
}
```

DAFNY: RESOURCES

- <https://rise4fun.com/Dafny/tutorial/Guide>
- <https://github.com/dafny-lang/dafny>